Kreiranje Tkinter grafičke aplikacije

Iako je u prethodnoj lekciji već bilo govora o prvoj Tkinter aplikaciji, u ovoj ćemo se detaljnije baviti Tkinter grafičkim okvirom i principima na kojima je on zasnovan.

Tkinter aplikacija se može kreirati koristeći funkcionalni i objektno orijentisani pristup. O funkcionalnom pristupu je već bilo reči, a sada ćemo pokazati kako se Tkinter aplikacija kreira na objektno orijentisani način.

Kako smo već rekli, za pokretanje Tkinter aplikacije neophodno je prvenstveno instancirati glavni prozor aplikacije. U OOP pristupu ćemo samo učiniti da naša klasa nasleđuje tk.Tk objekat, dok će samo instanciranje tog objekta biti izvršeno na sledeći način:

```
tk.Tk.__init__(self, *args, **kwargs)
```

Sada možemo dodati linije iz prethodnog primera vodeći računa o zameni root objekta ključnom rečju self:

OOP pristup Hello World primeru

```
import tkinter as tk
class HelloWorld(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.minsize(200, 50)
        Label(self, text="Hello World!").pack()

HelloWorld().mainloop()
```

Frame kontrola

Jedna od značajnijih kontrola u Tkinteru je Frame kontrola, koja nam omogućava grupisanje drugih kontrola, kao što su kontrole za unos teksta, labele, liste itd., što će nam biti korisno prilikom korišćenja organizatora kontrola (geometry managers), o kojima će biti reči nešto kasnije. Ako koristimo frejm, možemo podeliti aplikaciju na više različitih celina, pa je čest slučaj u praksi da se kreira čitava klasa koja će predstavljati funkcionalnost objedinjenu jednim frejmom. Ako bismo pratili MVC šablon kreiranja aplikacija, na ovaj način bismo kreirali view komponentu MVC šablona, odnosno – prikaz. Iskoristićemo klasu iz prethodnog primera, s tim što ćemo labelu prebaciti u klasu HelloFrame koja će predstavljati Frame objekat koji sadrži upravo tu labelu:

Pomoćna Tkinter klasa aplikacije sa Frame objektom

```
import tkinter as tk

class HelloFrame(tk.Frame):
    def __init__(self):
        tk.Frame.__init__(self)
        label = tk.Label(self, text="Hello world!",
        font=("Verdana", 10)).pack(pady=10, padx=10)
```

Copyright © Link group

Pošto je reč o Frame objektu, naša klasa mora naslediti tk.Frame klasu, kao i instancirati tk.Frame objekat metodom __init__. U ovu klasu smo sada prebacili našu labelu i izmenili smo tu labelu. Prvo smo definisali font u labeli ključnim argumentom font, kojem se prosleđuje n-torka sa dva elementa: imenom fonta i veličinom fonta. (Lista fontova se može proveriti sledećim kodom: from tkinter import Tk, font; root = Tk(); font.families()). Takođe, prilikom pakovanja ove kontrole koristili smo padx i pady ključne argumente koji se odnose na padding – razmak između naše kontrole i okolnih elemenata; u ovom slučaju je to ivica frejma.

Sada HelloFrame klasa tipa Frame sadrži željene kontrole i možemo je pozvati u HelloWorld klasi:

Glavna Tkinter klasa aplikacije

```
import tkinter as tk

class HelloFrame(tk.Frame):
    def __init__(self):
        tk.Frame.__init__(self)
        label = tk.Label(self, text="Hello world!",
        font=("Verdana", 10)).pack(pady=10, padx=10)



class HelloWorld(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.minsize(200, 50)
        HelloFrame().pack()

HelloWorld().mainloop()
```

Iz tog primera vidimo da se ostatak HelloWorld klase nije promenio, osim što smo instancirali HelloFrame objekat i pozvali .pack() metodu kako bi se taj Frame objekat i prikazao u glavnom prozoru.

Na kraju, ne treba zaboraviti da je potrebno instancirati HelloWorld klasu i staviti je u beskonačnu petlju metodom mainloop(). Ono što ova metoda čini jeste da *blokira* naš program, odnosno, ne dozvoljava da nam se program završi, već ga drži u beskonačnoj petlji i na taj način održava i prozor aplikacije na ekranu. U petlji koja je implementirana unutar mainloop()metode se izvršava osluškivanje na događaje od kojih jedan može biti i gašenje aplikacije koje korisnik ili kod iniciraju. Kada se upravo taj događaj desi, petlja gasi naš program, odnosno *odblokira* Python skriptu.

Treba napomenuti da ćemo dobiti gotovo vizuelno identičnu aplikaciju kao i u prvom primeru, ali nam ovakav pristup omogućava veću fleksibilnost kada se od nas zahteva kreiranje mnogo kompleksnije GUI aplikacije. Takođe, koristeći ovaj pristup moguće je napraviti i više različitih klasa koje nasleđuju Frame objekat, što je idealan pristup u situaciji gde se od programera traži aplikacija sa više stranica na kojima se nalaze različite funkcionalnosti.

Copyright © Link group

Iako to nismo koristili u ovom primeru, sam Frame objekat podržava određeni set opcija koje mu se mogu proslediti. Za detaljniju listu ovih opcija i njihova objašnjenja možete posetiti ovaj link.

Izmenite kod tako što ćete napraviti prozor dimenzije 300x100 u kome treba da se nalazi labela sa tekstom BIG LABEL definisantim fontom Impact veličine 20. Labela treba da bude udajena za 90 od gornje ivice.

Tkinter promenljive

Iako Python sadrži sasvim dovoljno tipova promenljivih, Tkinter nam nudi sopstvene promenljive, čija se upotreba razlikuje od upotrebe Pythonovih. Reč je o posebnim objektima koji, pored toga što sadrže određeni tip podatka, sadrže i specijalnu funkcionalnost koja se zasniva na automatskom *obaveštavanju* našeg programa kad god se vrednost u tim promenljivama promeni. Ovo znači da, ako imamo promenljivu a koju smo povezali sa dve različite labele, kad god bi se desila promena vrednosti te promenljive, bilo kad i bilo gde u programu – ta promena bi se odmah prikazala i u tim kontrolama. Tkinter promenljivama se vrednost podešava metodom set(), a dobavlja metodom 'get()'. Postoje četiri tipa Tkinter varijabli:

- StringVar() – odnosi se na tekstualni tip;
- IntVar() – odnosi se na cele brojeve;
- DoubleVar() – odnosi se na vrednosti sa pokretnim zarezom;
- BooleanVar() – odnosi se na binarne vrednosti (True/False 0/1).

Takođe, svakom od ovih konstruktora je moguće proslediti dva ključna argumenta: name i value, gde prvim argumentom možemo dodatno nazvati našu promenljivu, a argumentom value označavamo prvobitnu vrednost te promenljive. Dakle, ako želimo da dodelimo vrednost promenljivoj odmah prilikom instanciranja, možemo je definisati ovako: my_var = StringVar(value= 'Insert text here..'), a ako želimo da promenimo vrednost promenljive u toku programa, iskoristićemo sledeću naredbu:

```
my_var.set("Insert text here!")
```

Kako bismo pokazali funkcionalnost Tkinter promenljivih gde se promena promenljive na jednom mestu propagira kroz čitavu aplikaciju, iskoristićemo HelloFrame klasu iz prethodnog primera. Naime, želimo da dodamo novu kontrolu po imenu Entry, koja služi za to da korisnik prihvati tekst. Limit ove kontrole je to što može da prihvati samo jednu liniju teksta. Ako želimo da omogućimo korisniku da unose tekst sa više linija, možemo iskoristiti kontrolu Text(). Nakon što korisnik unese tekst u Entry kontrolu za koju će biti vezana StringVar() promenljiva (dakle, nakon što je izmeni), želimo da prikažemo promenu te promenljive u labeli. Kod HelloFrame klase izgleda ovako:

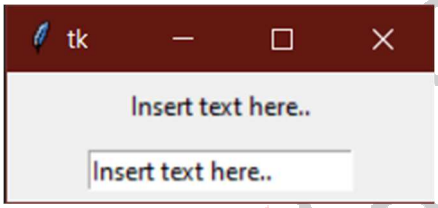
Primer korišćenja StringVar promenljive

```
import tkinter as tk

class HelloFrame(tk.Frame):
    def __init__(self):
        tk.Frame.__init__(self)
        my_var = tk.StringVar(value = 'Insert text here...')
```

Copyright © Link group

```
        my_var.set("Insert text here!!!")
        tk.Label(self, textvariable = my_var).pack(padx = 5, pady = 5)
        tk.Entry(self, textvariable = my_var).pack(padx = 5, pady = 5)

class HelloWorld(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.minsize(300, 200)
        HelloFrame().pack()

HelloWorld().mainloop()
```

U poređenju ove klase sa prethodnom verzijom vidimo promenu u instanciranju labele. Reč je o korišćenju ključnog argumenta textvariable umesto text kojim smo prethodno unosili vrednost. Upravo nam ključni argument textvariable omogućava propagaciju promene promenljive kroz program, što možemo i videti ako ga pokrenemo:



Slika 3.1. Izgled primera sa StringVar promenljivom

Prilikom pokretanja vidimo tekst na koji smo inicijalno podesili našu my_var promenljivu. Ako bismo kliknuli u donju kontrolu (Entry kontrola) i počeli da unosimo tekst – tekst u labeli bi se takođe simulatno menjao.

Izmenite kod tako što ćete postaviti da padding odn. vertikalna udaljenost labele bude 50 a polja za unos 90. Podesiti da se tekst polja bude prazan string a labele „Unesite tekst". Takođe promenite dimenziju ekrana u 400x400.

Callback i trace funkcionalnost promenljivih

Kako nam Tkinter omogućava praćenje i ažuriranje promena promenljivih kroz kontrole, takođe nam i nudi dodatnu funkcionalnost u vidu metode trace, pomoću koje možemo implementirati dodatnu sopstvenu logiku koja će se izvršavati pri promeni vrednosti. Metode Tkinter promenljivih koje nam ovo nude su:

- trace_add(mode, callback_func_name) – nad promenljivom registruje prosleđenu callback funkciju (u ovom slučaju je to callback_func_name) koja će se izvršiti kad god program pristupa promenljivoj;
- trace_remove(mode, callback_func_name) – nad promenljivom odjavljuje callback funkciju koja je prethodno bila postavljena trace_add() metodom;
- trace_info() – nad datom promenljivom pruža informaciju o callback funkciji.

Copyright © Link group

Argumenti za trace_add() i trace_remove() su isti, a prvi argument predstavlja mod koji definiše okolnosti u kojima će se prosleđena callback funkcija izvršiti (u ovom slučaju je to callback_func_name). Lista modova je array, read, write i unset ili lista/n-torka čiji su elementi neki od pomenutih modova. Više o ovim modovima se može videti u zvaničnoj dokumentaciji.

Pa tako, ako želimo da se naša funkcija callback_func_name() izvrši kad god program iščita promenljivu my_var – koristićemo read mod. Ova funkcionalnost je prikazana na sledećem primeru, koji je baziran na HelloFrame klasi (HelloWorld klasa ostaje ista):

Primer korišćenja callback funkcije

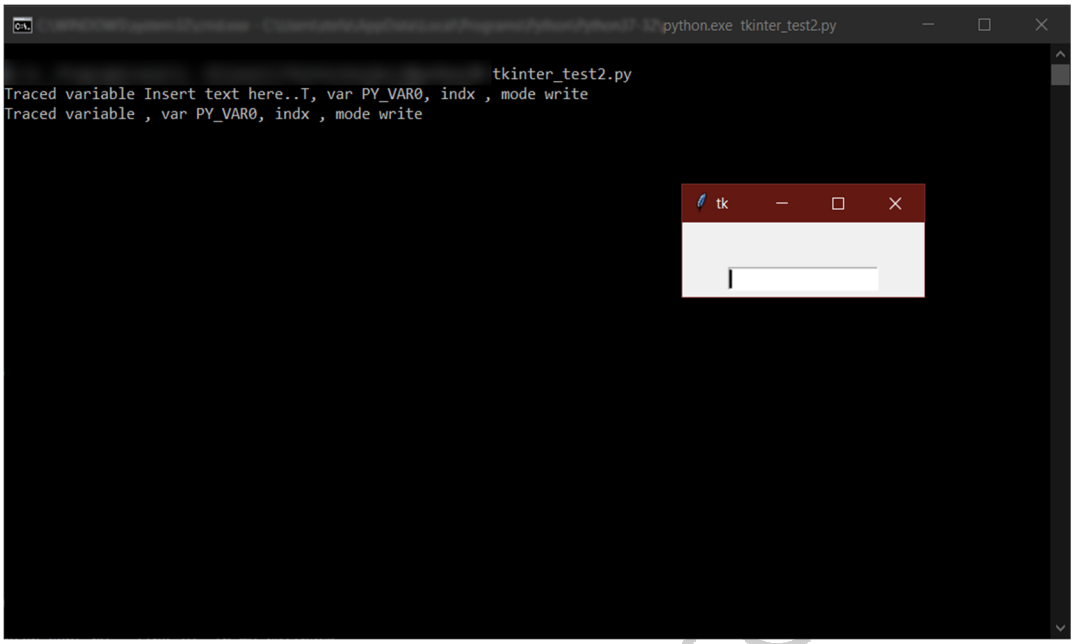
```
import tkinter as tk
class HelloFrame(tk.Frame):
    def __init__(self): # self, parent
        tk.Frame.__init__(self) # self, parent
        my_var = tk.StringVar(value = 'Insert text here..')

        def my_callback(var_name, indx, mode):
            print("Traced variable {}, var{}, indx{}, mode
{}".format(my_var.get(), var_name, indx, mode))

        my_var.trace_add('write', my_callback)
        # my_var.set("Insert text here..")
        tk.Label(self, textvariable = my_var).pack(padx = 5, pady = 5)
        tk.Entry(self, textvariable = my_var).pack(padx = 5, pady = 5)
class HelloWorld(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.minsize(300, 200)
        HelloFrame().pack()

HelloWorld().mainloop()
```

Ono što je takođe bitno prilikom definisanja callback funkcije su argumenti koja ona mora primati, kojih ima tri:

- ime promenljive nad kojom se callback funkcija izvršava (u našem primeru var_name argument). Ako ime promenljive nije postavljeno ključnim argumentom name prilikom instanciranja promenljive, Python prevodilac će sam dodeliti ime;
- indeks promenljive nad kojom se callback funkcija izvršava (u našem primeru argument indx);
- mod promenljive koji govori o tome u kom slučaju će se izvršiti callback funkcija (u našem primeru je to argument mode).

Nakon pokretanja ove aplikacije u komandnom prozoru odakle smo pokrenuli skript, a nakon što počnemo da menjamo tekst u Entry kontroli – primetićemo ispis koji je rezultat my_callback funkcije koja se izvršava jer program koristi write metodu nad my_var promenljivom:

Copyright © Link group



Slika 3.2. Izgled primera sa callback funkcijom

Izmenite kod tako da je tekst polja i labele bude prazan string pomoću set metode na početku programa. Kada se upiše tekst u polje potrebno je da se ispiše u komandom prozoru, dok tekst labele je potrebno da ostane prazan string.

Organizatori kontrola (geometry managers)

U prethodnim primerima smo koristili metodu pack() kojom smo željeni vidžet prikazivali na ekranu. Ta metoda je jedan od tri organizatora kontrola u Tkinteru. Ostala dva su metode grid() i place(). Sve tri imaju svoje prednosti i mane i odgovaraju različitim situacijama:

- pack() – koristimo kada želimo da naslažemo kontrole ili jednu na drugu ili jednu pored druge;
- grid() – koristi se za kreiranje mreže – tabele sa kolonama i redovima gde se kontrole mogu smeštati;
- place() – omogućava precizno postavljanje kontrola na prozoru zadavanjem tačnih koordinata mesta na kojem želimo da se kontrola nađe.

Preporučljivo je da se koristi samo jedan organizator nad kontejnerskom kontrolom, dok različite kontejnerske kontrole mogu koristiti i različite organizatore.

Metoda pack()

Iako smo ovu metodu koristili i pre, pokazaćemo sada koje još dodatne opcije ona podržava i kako se one odražavaju na kontrole nad kojim se metoda poziva. Za primere ćemo iskoristiti jednostavan kod koji nije baziran na OOP pristupu:

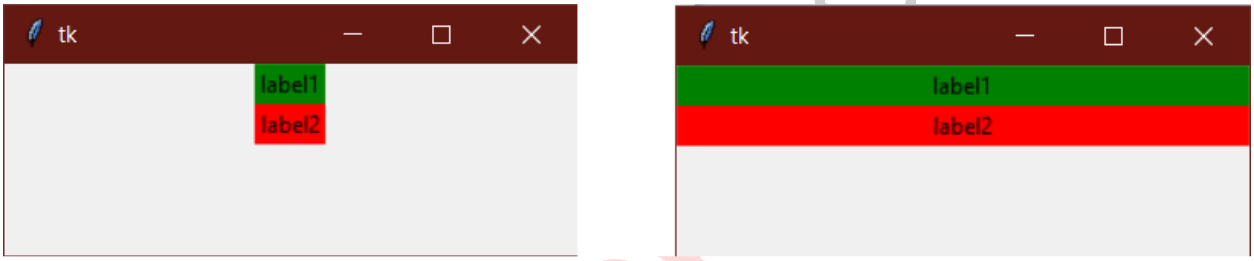
Copyright © Link group

Primer korišćenja pack() metode

```
import tkinter as tk
win = tk.Tk()
win.minsize(300, 100)
tk.Label(win,text = 'label1', bg = 'green').pack()
tk.Label(win,text = 'label2',bg = 'red').pack()
win.mainloop()
```

Najbitnije opcije koje možemo proslediti pack() metodi su:

- fill – prima tri različite opcije kao vrednosti: tk.X, tk.Y i tk.BOTH. Ove opcije predstavljaju ose po kojima možemo naglasiti kontroli da popuni ostatak slobodnog mesta na dodeljenoj osi. Na slici 3.3. prikazan je primer rasporeda labela sa korišćenjem argumenta fill = tk.BOTH u pack() metodi i bez korišćenja ovog argumenta:



Slika 3.3. Izgled bez korišćenja opcije tk.BOTH (levo) i nakon korišćenja te opcije (desno)

Ako bi samo prva labela zauzimala celu širinu prozora kod bi izgledao:

```
import tkinter as tk
win = tk.Tk()
win.minsize(300, 100)
tk.Label(win,text = 'label1', bg = 'green').pack()
tk.Label(win,text = 'label2',bg = 'red').pack(fill = tk.BOTH)
win.mainloop()
```

Na levoj strani vidimo kako su labele spakovane po podrazumevanoj konfiguraciji, a na desnoj vidimo da su nakon prosleđivanja argumenta fill = tk.BOTH labele horizontalno zauzele sav dostupni prostor.

- expand – vrednosti su True/False; označava da li će vidžet ispuniti bilo koji slobodni prostor kontejnerskog vidžeta (parent vidžet) (u našem primeru parent vidžet je win objekat koji ujedno predstavlja i prozor aplikacije; umesto njega mogli smo kreirati Frame objekat i u njega smeštati kontrole). Podrazumevana vrednost je False. Ako bismo za obe labele prosledili expand = True, fill = tk.BOTH argumente (uz nadovezivanje, na primer, sa fill argumentom), omogućili bismo da naše labele zauzimaju čitav prozor – iako same labele imaju širinu i visinu, naredbom expand se proširuju koliko god kontejnerski objekat labele – u ovom primeru prozor aplikacije – nudi mesta:

Copyright © Link group

Slika 3.4. Primer korišćenja expand argumenta

Ako bismo samo na drugoj labeli ostavili expand = False – dobili bismo situaciju gde ta druga labela zauzima mesta tek koliko joj je potrebno, a prva labela zauzima sav ostali slobodni prostor:

Slika 3.5. Primer korišćenja expand argumenta na prvoj kontroli

- side – označava na kojoj strani u okviru kontejnerskog vidžeta će se vidžet nad kojim se poziva side() fiksirati. Podrazumevana vrednost je tk.TOP, ali postoje još i tk.BOTTOM, tk. LEFT i tk.RIGHT. Kao što smo rekli, podrazumevana vrednost ovog argumenta je tk.TOP; zato su se labele u dosadašnjim primerima slagale jedna na drugu. Ako bismo metodi pack() prosledili i side argument, ali sa parametrom tk.LEFT (za obe labele), dobili bismo sledeći raspored:

Slika 3.6. Primer korišćenja side argumenta

Isti izgled bismo dobili i da smo u jednoj labeli koristili tk.LEFT, a u drugoj tk.RIGHT, jer je reč o istoj osi. Da smo u samo jednoj od labela ostavili tk.TOP ili tk.BOTTOM – ne bismo dobili ovako ravnomerno postavljene kontrole. Takođe, ako želimo da nam prva labela zauzme čitav red, a da naredna labela zauzme samo levu ivicu, pozvaćemo pack() metodu na sledeći način:

Copyright © Link group

```
import tkinter as tk
win = tk.Tk()
win.minsize(300, 100)
tk.Label(win,text = 'label1', bg = 'green').pack(side = tk.TOP,  fill =
tk.BOTH)
tk.Label(win,text = 'label2',bg = 'red').pack(side = tk.LEFT, fill = tk.BOTH)
win.mainloop()
```

Slika 3.7. Primer kombinacije tk.TOP i tk.LEFT vrednosti

Takođe, pored ovih argumenata, u pack() metodi možemo koristiti i padx/pady (padding izvan ivica kontrole), ipadx/ipady (padding unutar ivica kontrole) i ostale opcije koje smo pominjali do sada.

Pitanje

Koja od sledećih metoda nije deo oganizatora kontrola?

- place()
- grid()
- **set()**

Objašnjenje:

Tačan odgovor je da metoda set() nije deo organizatora kontrola, već se koristi zajedno sa Tkinter promenljivama za postavljanje vrednosti.

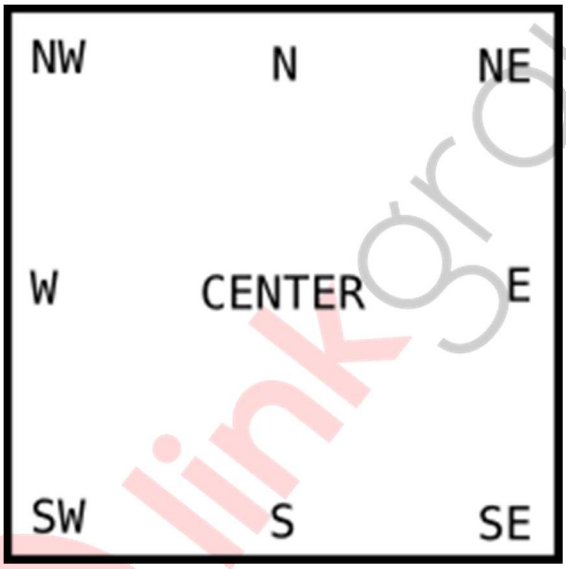
Metoda grid()

Pomoću ove metode kreiramo, takoreći, matricu, odnosno, kontejnerski objekat delimo na kolone i redove kojima pristupamo pomoću indeksa kolone i reda, slično kao i u dvodimenzionalnim listama, što nam omogućava rad sa strukturom nalik na tabelu. Prilikom poziva nad vidžetom, kao i kod pack() metode, imamo različite opcije koje možemo upotrebiti. Neke od bitnijih su:

- column – kolona u koju smeštamo vidžet smeštamo; podrazumevana vrednost je 0, što predstavlja prvu kolonu sa leve strane;
- columnspan – kroz koliko kolona će se vidžet prostirati; podrazumevana vrednost je 1;

Copyright © Link group

- row – red u koji smeštamo vidžet; podrazumevana vrednost je 0, što predstavlja prvi red odozgo;
- rowspan – kroz koliko redova će se vidžet prostirati; podrazumevana vrednost je 1;
- sticky – ako je vidžet koji smeštamo u polje naše *tabele* manji od tog polja, možemo nagovestiti u koji ugao tog polja želimo da *prilepimo* kontrolu. Vrednosti ovog argumenta su zapravo strane kompasa prosleđene kao stringovi ili kao predefinisane vrednosti iz tkinter modula: tk.E, tk.W, tk.S, tk.N ili 'W', 'E', 'S', 'N'. Takođe je moguće proslediti kombinaciju ovih strana:
    - ako smo se odlučili za prosleđivanje strana kompasa kao stringova, konkatenacijom – 'NS', 'SW', 'NSEW';
    - ako smo se odlučili za prosleđivanje preko predefinisanih vrednosti iz tkinter modula, n-torkama: (tk.NS), (tk.SW), (tk.NSEW).



Slika 3.8. Slika sa orijentacijama i mogućim kombinacijama orijentacija sticky argumenta[1]

Pre nego što pokažemo kako funkcioniše grid() metoda, možemo u vidžetu koji će sadržati naše labele (u našem primeru je to glavni prozor aplikacije, ali može biti i npr. Frame) konfigurisati tu svojevrsnu tabelu u čija ćemo polja smeštati labele (ovo nije neophodno, ali ako se ne odradi, ta *tabela* će koristiti predefinisane podrazumevane vrednosti za kolone i redove). Tako, naš osnovni primer izgleda ovako:

[1] https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/anchors.html

Copyright © Link group

Primer korišćenja grid() metode

```
import tkinter as tk

win = tk.Tk()
win.minsize(300, 100)

win.columnconfigure(0, weight=1)
win.columnconfigure(1, weight=2)

l1=tk.Label(win,text = 'label1', bg = 'green')
l1.grid(column=0,row=0)
l2=tk.Label(win,text = 'label2',bg = 'red')
l2.grid(column=0,row=1)
win.mainloop()
```

Konfigurisanje kontejnerske kontrole za rad sa grid() organizatorom postižemo koristeći columnconfigure() i rowconfigure() metode. U ovom slučaju smo odlučili da konfigurišemo samo kolone. Obe metode funkcionišu identično, gde je prvi argument indeks kolone ili reda koji konfigurišemo, a parametar weight označava koliko će prostora ta kolona zauzimati. U našem primeru smo postavili da druga kolona (indeks 1) bude duplo veća nego prva. Iako smo u primeru konfigurisali dve kolone, moguće je imati ih i više. Takođe je moguće odjednom konfigurisati više kolona/redova, tako što će se njihovi indeksi proslediti kao n-torka:

```
win.columnconfigure((0,2,3,4), weight=1)
```

Ako bismo odmah pokrenuli naš primer, dobili bismo ovakav rezultat:

Slika 3.9. Dodavanje kontrola u istu kolonu

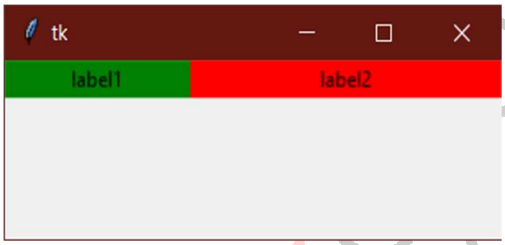
Ovo možda nije rezultat koji smo očekivali, ali Tkinter, odnosno grid() metoda svaku novu kontrolu smešta u sledeći red prve kolone (indeks 0). Kako bismo postavili labele tako da je svaka u svojoj koloni u istom redu, moramo obaviti sledeći grid() poziv (koristimo row/column argumente):

```
tk.Label(win,text = 'label1', bg = 'green').grid(column = 0, row = 0)
tk.Label(win,text = 'label2',bg = 'red').grid(column = 1, row = 0)
```

Copyright © Link group

Slika 3.10. Dodavanje kontrola u isti red

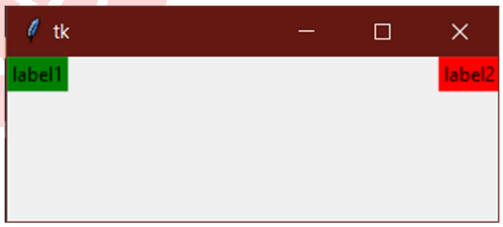
Ovo za sada izgleda u redu, ali još uvek, na primer, ne vidimo ivice kolona, odnosno koliko se prvi red u obe kolone prostire. Pošto naše labele imaju pozadinsku boju, ovo možemo videti koristeći sticky argument grid()metode. Kao vrednost prosledićemo sve strane sveta, odnosno string NSEW za obe labele, i time raširiti naše labele tako da zauzimaju sav slobodan prostor u svom polju:



Slika 3.11. Proširivanje labela tako da zauzmu čitav red u koloni

Sada možemo uočiti svojstvo druge kolone s početka ovog primera, a to je da je ta kolona duplo šira nego prva (argument weight). Ako bismo želeli samo da *zadržimo* ove labele tako da je svaka u svom uglu, prvoj labeli bismo prosledili W vrednost sticky argumenta, a drugoj labeli vrednost E:

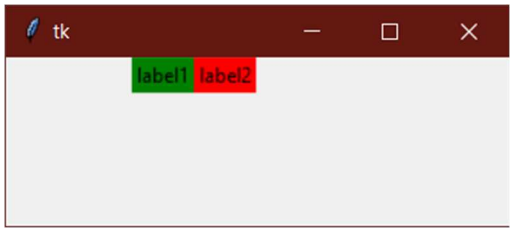
```
tk.Label(win,text = 'label1', bg = 'green').grid(column = 0, row = 0, sticky
= 'W')
tk.Label(win,text = 'label2',bg = 'red').grid(column = 1, row = 0, sticky =
'E')
```



Slika 3.12. Postavljanje labela uz ivice glavnog prozora

U ovom primeru je sada svaka labela na svojoj strani (prva na levoj, druga na desnoj strani prvog reda) i u svojoj koloni. Ako bismo okrenuli i prvoj labeli, argumentu sticky prosledili E, a drugoj labeli prosledili W, dobili bismo situaciju gde su labele jedna do druge:

Copyright © Link group



Slika 3.13. Postavljanje labela jedne uz drugu (zajednička ivica kolona)

Ako bismo konfigurisali obe kolone sa weight parametrom kao 1, dobili bismo situaciju gde su širine kolona identične, a granica između njih na sredini prozora aplikacije. Takođe je moguće kreirati i prazne redove/kolone koji nisu u upotrebi od strane kontrola, ali takođe mogu uticati na raspored kontrola.

Metoda place()

Place metoda nam omogućava da postavimo poziciju i veličinu vidžeta koristeći relativne ili apsolutne vrednosti. U poređenju sa pack() i grid() metodama, retko se koristi, ali može poslužiti u nekim kompleksnim scenarijama dizajniranja GUI-a. Argumenti koji se mogu proslediti su:

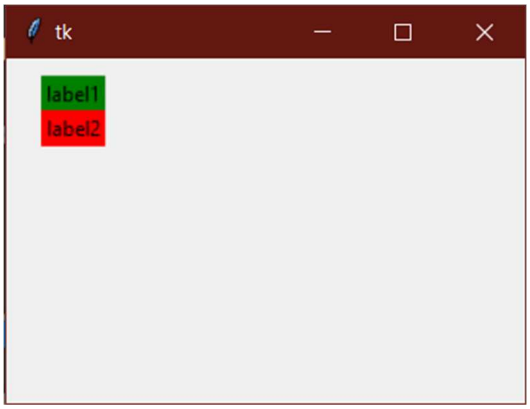
- anchor – reč je o sličnoj funkcionalnosti kao i kod sticky argumenta grid() metode, pa se tako mogu proslediti i iste vrednosti; ovim argumentom označavamo na kojoj ćemo strani kontejnerskog vidžeta postaviti svoj vidžet;
- bordermode – postoje dve vrednosti ovog argumenta koje možemo koristiti: INSIDE i OUTSIDE. Ovaj argument se odnosi na ignorisanje ivica kontejnerskog vidžeta;
- height/width – označava vrednost visine/širine vidžeta nad kojim se place() metoda poziva; ove vrednosti su u pikselima i tipa su int;
- relheight/relwidth – ovim parametrima podešavamo relativnu visinu/širinu u odnosu na visinu/širinu kontejnerskog vidžeta. Vrednost je decimalna u opsegu od 0.0 do 1.0;
- relx/rely – ovim parametrima podešavamo horizontalni/vertikalni pomeraj vidžeta u odnosu na visinu/širinu kontejnerskog vidžeta. Takođe su dozvoljene vrednosti u opsegu 0.0–1.0;
- x/y – ovim parametrom podešavamo horizontalne/vertikalne koordinate mesta gde će se prikazati naša kontrola.

Primer apsolutnog pozicioniranja gde ćemo postaviti labele jednu ispod druge:

Primer korišćenja place() metode

```
import tkinter as tk
win = tk.Tk()
win.minsize(300, 100)
tk.Label(win,text = 'label1', bg = 'green').place(x = 20, y = 10)
tk.Label(win,text = 'label2',bg = 'red').place(x = 20, y = 30)
win.mainloop()
```

Copyright © Link group



Slika 3.14. Korišćenje place() metode

Kao što vidimo, dimenzije našeg prozora su 300x100 piksela. Ako bismo, na primer, postavili bilo koju kontrolu izvan tih dimenzija, mogli bismo da ih vidimo samo ako možemo da proširimo naš prozor; u suprotnom, neće biti vidljive. Takođe je moguće postaviti kontrolu iznad kontrole i u tom scenariju će se videti poslednja postavljena kontrola, dok ostale kontrole, koje su ispod, neće biti ni vidljive ni dostupne za korišćenje.

Izmenite kod tako da dimenzija prozora bude 30x300. Podesite koordinate tako da se zelena labele nalazi u gornjem levom uglu a crvena u gornjem desnom uglu.

Rezime

- Tkinter aplikacija se može kreirati koristeći funkcionalni i objektno orijentisani pristup.
- Jedna od značajnijih kontrola u Tkinteru je Frame, koja nam omogućava grupisanje drugih kontrola kao što su kontrole za unos teksta, labele, liste itd., što je korisno prilikom korišćenja organizatora kontrola (geometry managers).
- Sa frejmom, možemo podeliti aplikaciju na više različitih celina, pa je čest slučaj u praksi da se kreira čitava klasa koja će predstavljati funkcionalnost objedinjenu jednim frejmom.
- Vrste Tkinter promenljivih su: StringVar – za rad sa stringovima, IntVar – za rad sa celim brojevima, DoubleVar – za rad sa vrednostima sa pokretnim zarezom i BooleanVar – odnosi se na True/False vrednosti.
- Metode za rad sa callback funkcijama nad Tkinter promenljivama su: trace_add(), trace_remove() i trace_info().
- Organizatori kontrola su metode pack(), grid() i place().
- Metodom pack(), kontrole dodajemo jednu ispod druge.
- Pomoću grid(), praktično kreiramo matricu, odnosno, delimo kontejnerski objekat na kolone i redove kojima pristupamo pomoću indeksa kolone i reda, slično kao i u dvodimenzionalnim listama, što nam omogućava rad sa strukturom nalik na tabelu.
- Metoda place() nam omogućava da postavimo poziciju i veličinu vidžeta koristeći relativne ili apsolutne vrednosti.

Copyright © Link group